4Л

УДК 004.932.2:004.383.5

Ю.В. Ладиженський, А.О. Середа
Донецький національний технічний університет, Україна
ly@cs.dgtu.donetsk.ua, aas11@bk.ru

Пошук фрагментів зображень за шаблоном для відстежування об'єктів у відео на паралельній архітектурі CUDA

Розглянуто пошук фрагментів зображень довільної форми в кадрі за шаблоном методом повного перебору із використанням паралельної обчислювальної архітектури CUDA. Запропоновано і проаналізовано різні способи кешування області пошуку в пам'яті мультипроцесора. Експериментально оцінено прискорення порівняно з центральним процесором. Запропоновані алгоритми можуть використовуватись для прискорення відстежування об'єктів у відео.

Вступ

Необхідність відстежування об'єктів у відеопотоці виникає в системах безпеки, автоматизації аналізу спортивних змагань, контролю технологічних процесів, при організації людино-машинного інтерфейсу. Під відстежуванням розуміється отримання множини видимих об'єктів і їх координат у кожному кадрі.

Пошук зображень за шаблоном є одним із методів відстежування об'єктів у відео. У [1] шаблон цілого об'єкта представлений зображенням об'єкта і матрицею, що задає ймовірність приналежності до об'єкта кожного пікселя цього зображення. У [2] описано метод відстежування об'єктів у відеопотоці на основі відстежування переміщення фрагментів об'єктів, які представлені шаблонами. Близька задача пошуку прямокутних блоків зображення у кадрі виникає при кодуванні відео [3].

Одним із методів пошуку фрагмента зображення у кадрі є метод Lucas-Kanade tracker, який базується на обчислені градієнтна зображення і переміщенні у напрямку зменшення функції різниці між кадром та шаблоном [4]. Іншим підходом є безпосереднє порівняння шаблона з областями кадру [3], [5], яке звичайно використовується при кодуванні відео.

Пошук фрагмента зображення в кадрі може займати багато часу і потребує прискорення. Пошук складається з однотипних операцій над елементами матриць і може бути ефективно реалізований на паралельній обчислювальній архітектурі. Можна використовувати як спеціалізовані обчислювальні структури для пошуку фрагментів зображень в кадрі [5], так і паралельну архітектуру загального призначення. На даний час поширеними і доступними пристроями, придатними для паралельної обробки зображень і відео, є відеокарти NVIDIA з підтримкою технології CUDA [6].

Хоча існують послідовні алгоритми пошуку шаблонів [3], [4] та паралельні реалізації алгоритмів пошуку блоків при кодуванні відео [3], [5], не існує паралельних алгоритмів пошуку шаблонів зображень довільної форми, які можуть бути використані у методі відстежування об'єктів [2].

Метою даної роботи є розробка нових алгоритмів пошуку шаблонів зображень довільної форми у кадрі для архітектури CUDA. У даній статті розглядається пошук шаблону методом повного перебору.

Постановка задачі

Нехай фрагмент зображення, який шукають, вписано у прямокутник завширшки W та заввишки H. Нехай F – матриця розміром $H \times W$, що містить пікселі шаблону. Кожен піксель шаблону $f_{y,x}$ $(y = \overline{0, H-1}, x = \overline{0, W-1})$ характеризується наступними властивостями: $f^r_{y,x}$, $f^g_{y,x}$, $f^b_{y,x}$ – відповідно красна, зелена та синя компоненти кольору; $f^m_{y,x} \in [0,1]$ – оцінка приналежності пікселя з координатами (x, y) до фрагмента. Значення $f^m_{y,x}$ може бути чисельно не рівним ймовірності, але зі збільшенням ймовірності монотонно зростає.

Назвемо областю пошуку прямокутну область кадру, що містить пікселі шаблону при всіх його можливих положеннях. Нехай її висота дорівнює H_0, ширина – W_0, а C – матриця розміром $H_0 \times W_0$, що містить її пікселі. Кожен піксель області пошуку $c_{y,x}$ складається з $c^r_{y,x}$, $c^g_{y,x}$, $c^b_{y,x}$ – відповідно красної, зеленої та синьої компонент кольору.

Нехай осі системи координат кадру і шаблону спрямовані вправо і вниз, а піксель із координатами (0,0) розташовано в лівому верхньому куті.

Міру різниці між пікселем шаблону $f_{y,x}$ і пікселем області кадру $c_{y',x'}$ визначимо як

$$\left\| f_{x,y}, c_{y',x'} \right\| = f^m_{x,y} \left(\left| f^r_{x,y} - c^r_{y',x'} \right| + \left| f^g_{dx,dy} - c^g_{y',x'} \right| + \left| f^b_{dx,dy} - c^b_{y',x'} \right| \right). \quad (1)$$

Накладемо шаблон на кадр без повороту так, що піксель шаблону з координатами $(0,0)$ співпаде з пікселем кадру з координатами (x, y). Міру різниці між шаблоном і областю кадру під ним визначимо як

$$D(x, y) = \sum_{dx=0}^{W-1} \sum_{dy=0}^{H-1} \left\| f_{dx,dy}, c_{x+dx,y+dy} \right\|. \quad (2)$$

Нехай $S = \{(x_i, y_i)\}$ – множина всіх можливих координат лівого верхнього пікселя шаблону в системі координат області пошуку, причому

$$0 \le x_i \le W_s,\ 0 \le y_i \le H_s,\ W_s = W_0 - W + 1,\ H_s = H_0 - H + 1.$$

Найкращу позицію шаблону визначимо як $D_{best} = D(x_{best}, y_{best})$ і $(x_{best}, y_{best}) = \underset{(x,y)\in S}{\arg\min} D(x, y)$.

У методі відстежування [2] для визначення ймовірності успішного відстежування фрагмента зображення об'єкта необхідно також визначати альтернативну найкращу позицію фрагмента, тобто кращу з позицій, що залишилися в області пошуку, з якої вилучено околицю кращої позиції:

$$D_{alt} = \min_{(x,y)\in S, \min(|x - x_{best}|, |y - y_{best}|) \ge d} D(x, y),$$

де d – константа, що визначає розмір околиці, яка вилучається.

Для визначення (x_{best}, y_{best}) і D_{best} можна використовувати як повний перебір всіх можливих положень шаблону, так і швидкі алгоритми пошуку, засновані на монотонному зменшенні міри різниці між шаблоном і кадром при наближенні до оптимального рішення, такі, як Three-Step Search [3]. Для визначення D_{alt} доцільно використовувати алгоритм повного перебору. Пошук шаблонів повним перебором є найбільш ресурсоємним етапом відстежування об'єктів у методі [2] і потребує прискорення. Далі розглядаються його реалізації на архітектурі CUDA.

Відображення алгоритму пошуку шаблону зображення на архітектуру CUDA

Спрощена архітектура CUDA [6] наведена на рис. 1. Розглядаються пристрої з compute compatibility 1.1, як найпоширеніші на момент написання.

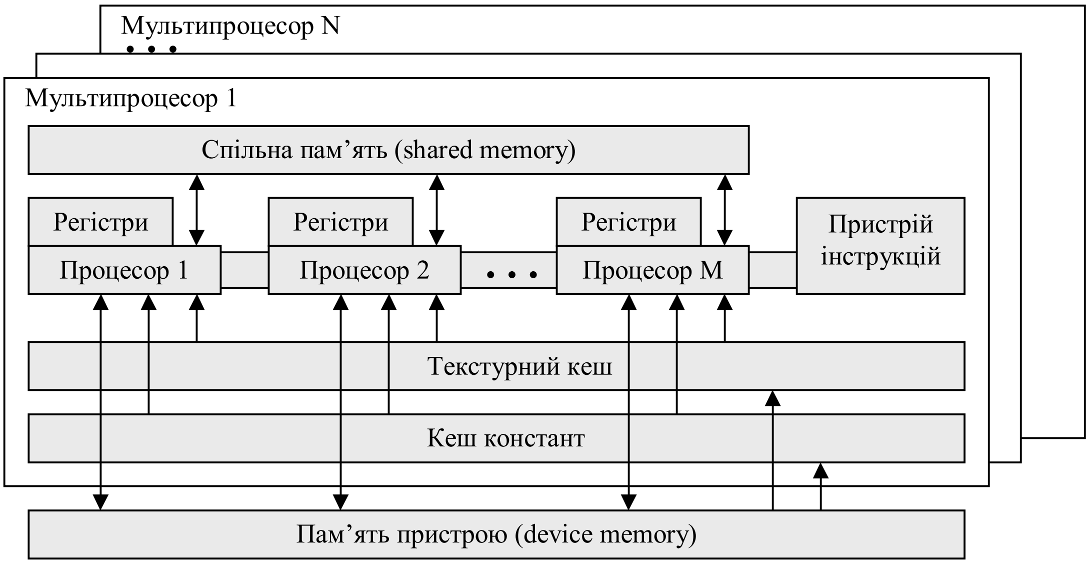


Рисунок 1 – Спрощена архітектура CUDA

На одному мультипроцесорі може одночасно виконуватись один або більше блоків потоків. Загальна кількість потоків на мультипроцесорі звичайно перевищує кількість скалярних процесорів, і групи потоків виконуються на них по черзі. Планувальник потоків оперує групами (warp) із 32 потоків одного блоку із послідовними номерами. Блоки, для виконання яких не вистачає вільних ресурсів, чекають у черзі, поки інші блоки не завершать роботу.

Для мінімізації обміну даними між пам'яттю пристрою та пам'яттю кожного мультипроцесора дані, що багато разів використовуються, доцільно читати один раз і поміщати в текстурний кеш або спільну пам'ять. Для завантаження кожного пікселя *F* і *C* з пам'яті пристрою мінімальне число раз, а також для усунення необхідності синхронізації і обміну даними між різними мультипроцесорами доцільно проводити пошук одного шаблону на одному мультипроцесорі, тобто використовувати для цього один блок потоків. Залежно від кількості потоків та спільної пам'яті в одному блоці мультипроцесор може одночасно шукати один чи більшу кількість шаблонів.

На сучасних пристроях цілочисельні арифметичні операції виконуються приблизно в чотири рази повільніше, ніж операції з плаваючою точкою [7]. Доцільно зберігати в пам'яті колір пікселів як цілі однобайтні числа, а перед обчисленнями перетворювати їх в дійсні числа.

При зверненні до пам'яті пристрою без використовування текстурного кешу бажано проводити з'єднані (coalesced) запити до пам'яті пристрою. Для цього 16 потоків, що виконуються одночасно, повинні звертатися до 16 чотирибайтних слів, розташованих в пам'яті послідовно з адреси, кратної 64 (деякі потоки можуть не брати участь).

Необхідність обчислювати D_{alt} спричиняє необхідність зберігати знайдені значення $D(x,y)$ для $(x,y)\in S$. Їх об'єм може бути завеликий для зберігання в пам'яті мультипроцесора, тому вони зберігаються в пам'яті пристрою. Щоб набувати готові

значення (2) в регістрах, кожне значення D повинне обчислюватися одним потоком. Зручно, щоб група з 16 потоків обчислювала 16 значень D, розташованих в пам'яті послідовно. Якщо потоків менше, ніж H_sW_s, то деякі потоки можуть обчислювати і зберігати більше одного значення D.

При запропонованій організації обчислень прискорення пошуку множини шаблонів досягається як за рахунок одночасного пошуку різних шаблонів на різних мультипроцесорах, так і за рахунок одночасного обчислення значень (2) різними потоками для різних позицій шаблону.

Загальний алгоритм пошуку шаблону

Розглянемо частину алгоритму пошуку шаблону, яка не залежить від способу кешування області пошуку.

Нехай є M потоків і i-й потік ($i = \overline{0, M-1}$) обробляє множину позицій шаблону $S^i \subset S$. Він обчислює і зберігає в пам'яті пристрою $D(x, y)$ для всіх $(x, y) \in S^i$. В процесі обчислення різних значень D кожен потік обчислює і зберігає у своїх регістрах значення $\left(x^i_{best}, y^i_{best}\right) = \arg\min\limits_{(x,y)\in S^i} D(x, y)$ і $D^i_{best} = D\left(x^i_{best}, y^i_{best}\right)$.

Після завершення обчислення $D(x, y)$ для $(x, y) \in S$, а також x^i_{best} , y^i_{best} і D^i_{best} , для $i = \overline{0, M-1}$, визначається такий номер потоку i_{best}, що $\forall_{i\in[0,M-1]} \left(D^i_{best} > D^{i_{best}}_{best}\right) \vee$ $\vee \left((D^i_{best} = D^{i_{best}}_{best}) \wedge (i \geq i_{best})\right)$. Потік i_{best} зберігає в змінних в спільній пам'яті значення $x_{best} = x^{i_{best}}_{best}$, $y_{best} = y^{i_{best}}_{best}$, $D_{best} = D^{i_{best}}_{best}$.

Після обчислення D_{best} кожен потік обчислює $D^i_{alt} = \min\limits_{(x,y)\in S^i_{alt}} D(x, y)$, де $S^i_{alt} = S^i \setminus \left\{(x, y) : \min\left(\left|x - x_{best}\right|, \left|y - y_{best}\right|\right) < d\right\}$. При цьому використовуються раніше збережені в пам'яті пристрою значення D. D_{alt} вибирається з $\left\{D^i_{alt}\right\}$ так само, як на попередньому кроці D_{best} з $\left\{D^i_{best}\right\}$. D_{best} і D_{alt} діляться на норму $\sum\limits_{x=0}^{W-1}\sum\limits_{y=0}^{H-1} f^m_{dx,dy}$.

Нижче розглянуто різні способи обчислення D залежно від способу кешування області пошуку. В описаних нижче алгоритмах пікселі шаблону можуть бути розташовані як у текстурному кеші, так і в спільній пам'яті.

Кешування області пошуку у текстурному кеші

Пропонується схема розподілу задач по потоках: i-й потік обчислює множину позицій $S^i = \left\{((x + i + jM) \bmod W_s, (y + i + jM) \operatorname{div} W_s)\right\}$ для $j = \overline{0, \left(H_sW_s + M - 1 - i\right) \operatorname{div} M}$, де div позначає цілочисельне ділення, а mod – залишок від ділення. Елементи масиву D розташовано в пам'яті послідовно по рядках без дір з адреси, кратної 64.

На рис. 2 показані три послідовні етапи пошуку шаблону розміром 3×3 пікселя в області пошуку розміром 7×11 пікселів з використанням 16 потоків. Заштрихованими квадратиками показані пікселі, з якими на кожному кроці збігається лівий верхній кут шаблону, а сірим – вже оброблені пікселі.

Псевдокод алгоритму обчислення x^i_{best} , y^i_{best} , D^i_{best} і масиву D показано на рис. 3. Алгоритм може працювати при будь-яких значеннях W_0, H_0, W та H і забезпечує повне завантаження всіх потоків при $H_sW_s \equiv 0 (\operatorname{mod} M)$. Усі звернення до пам'яті пристрою є з'єднаними.

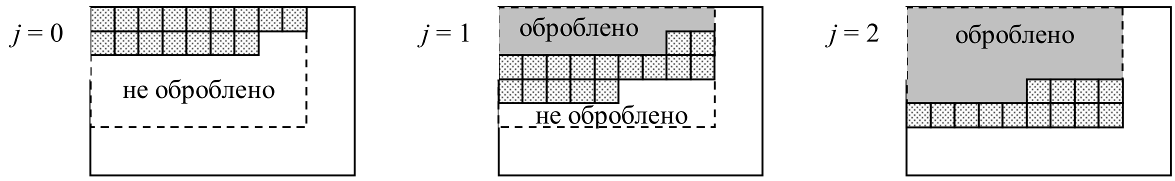


Рисунок 2 – Обробка області пошуку з використанням текстурного кешу

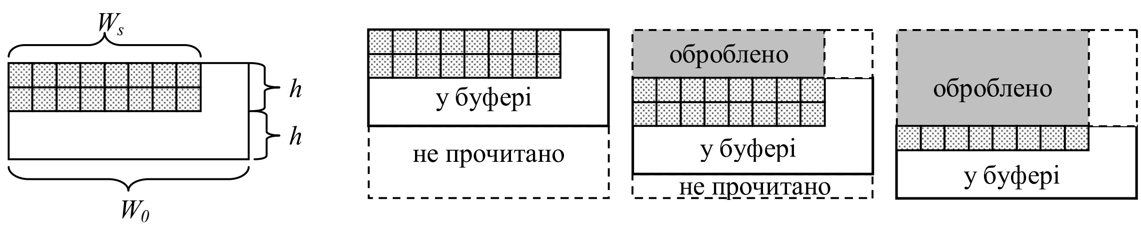
$D^{i}_{best} := \infty$; { підготовка до обчислень }
$k := i$;
поки $k < H_s W_s$ { цикл по позиціях шаблону }
  $x_0 := k \bmod W_s$; { визначення координат шаблону }
  $y_0 := k \operatorname{div} W_s$;
  $s := 0$; { обчислення $D(x_0, y_0)$ }
  для y від 0 до H-1
    для x від 0 до W-1
      $s := s + Dist(f[y,x], c[y_0 + y, x_0 + x])$;
  зберегти $D[y_0, x_0] = s$; { збереження результату }
  якщо $D^{i}_{best} > s$ { перевірка найкращого результату у потоці }
    $D^{i}_{best} := s$; $x^{i}_{best} := x_0$; $y^{i}_{best} := y_0$;
  $k := k + M$; { перехід до наступної позиції шаблону }

Рисунок 3 – Алгоритм пошуку шаблону при використанні текстурного кешу

Кешування області пошуку у спільній пам’яті

Пропонується читати рядки області пошуку зверху вниз і зберігати використовувані на кожному кроці рядки в циклічному буфері. Нехай на кожному кроці (окрім першого) в буфер завантажується h рядків, а після кожного читання перевіряється $W_s h$ позицій шаблону. Щоб на кожному кроці при обчисленні D (окрім, можливо, останнього кроку) були зайняті всі потоки, повинна виконуватись умова $W_s h \equiv 0 (\bmod M)$. Визначимо мінімальну достатню для обчислень кількість блоків заввишки h у буфері як $k = \lceil (h + H - 1)/h \rceil$. Кожен j-й рядок області пошуку зберігатиметься в рядку буфера із номером ($j \bmod kh$).

Наприклад, при $W = H = 3$, $W_0 = 10$, $H_0 = 7$ можна вибрати значення $h = 2$, $M = 16$ і $k = 2$ (такі малі значення звичайно не виникають в задачі відстежування об’єктів, не є оптимальними і наведені для пояснення алгоритму). На рис. 4 а) показаний циклічний буфер, на рис. 4 б) показані три послідовні етапи обробки області пошуку. Заштрихованими квадратиками показані пікселі, з якими поєднується лівий верхній кут шаблону.



а) б)

Рисунок 4 – Розбиття області пошуку по рядках

Для оптимальної роботи планувальника потоків CUDA повинна виконуватись умова $M \equiv 0(\bmod 32)$ [7], а значить, $W_s h \equiv 0(\bmod 32)$. Для досягнення повного завантаження потоків на останньому кроці повинна виконуватися умова $(H_0 - H + 1) \equiv 0(\bmod h)$. Читання кожного елементу області пошуку відбувається рівно один раз.

Описаний алгоритм можна узагальнити на випадок, коли спільної пам'яті мультипроцесора не вистачає для зберігання буфера заввишки kh і завширшки W_0.

Пропонується використовувати циклічний буфер заввишки kh і завширшки w, причому ($w \geq W$). Величини k и h визначаються так само, як в попередньому алгоритмі. На кожному кроці (окрім першого) в буфер завантажується h рядків, а після кожного читання перевіряється $(w - W + 1)h$ позицій шаблону. Кількість потоків є дільником $(w - W + 1)h$. Кожен j-й рядок області пошуку зберігається в рядку буфера $(j \bmod kh)$.

Область пошуку розбивається на стовпці завширшки $(w - W + 1)$. Кожен стовпець, окрім останнього, обробляється так само, як в попередньому алгоритмі. Оскільки ширина буфера перевищує ширину стовпця, при завантаженні в буфер рядків j-го стовпця в буфер також потрапляють деякі елементи $(j+1)$-го. Останній стовпець може мати ширину менше ніж $(w - W + 1)$, і він обробляється, тільки якщо його ширина не менше W.

Наприклад, при $W = H = 3$, $W_0 = 12$, $H_0 = 8$ можна вибрати значення $h = 3$, $w = 6$, $M = 8$ і $k = 2$. На рис. 5 а) показаний циклічний буфер, на рис. 5 б) показані три з дев'яти послідовних етапів обробки області пошуку при обраних параметрах.

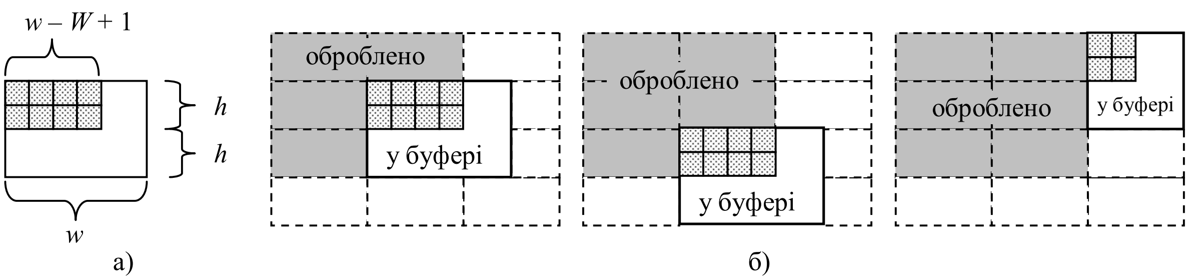


Рисунок 5 – Розбиття області пошуку по рядках і стовпцях

Для оптимальної роботи планувальника потоків CUDA повинна виконуватись умова $M \equiv 0(\bmod 32)$, а значить, $(w - W + 1)h \equiv 0(\bmod 32)$. Для досягнення повного завантаження потоків при обробці нижнього ряду і правого стовпця блоків повинні виконуватися умови $(H_0 - H + 1) \equiv 0(\bmod h)$ і $(W_0 - W + 1) \equiv 0(\bmod(w - W + 1))$.

При $w = W_0$ даний алгоритм виконує розбиття області тільки по рядках і поводиться аналогічно попередньому. При $w \neq W_0$ алгоритм проводить повторні читання з пам'яті пристрою деяких значень $c_{y,x}$: в середньому на кожні $(w - W + 1)$ пікселів алгоритм проводить w читань.

Псевдокод алгоритму, що проводить розбиття по рядках і стовпцях, приведений на рис. 6.

Не для всіх можливих значень W_0, H_0, W і H можливо вибрати значення h, k і M, що забезпечують оптимальне завантаження процесорів. У обох алгоритмах при будь-яких розмірах шаблону і області пошуку можливо зробити всі запити до пам'яті пристрою з'єднаними.

$D_{best}^i := \infty$; { підготовка до обчислень }
$x_1 := 0$; { (x_1, y_1) – координати лівого верхнього кута частини області пошуку, що кешується }
поки $x_1 < W_s$ { цикл по стовпцях області пошуку }
    $j := i$;
    поки $j < wh$ {завантаження верхньої частини стовпця}
        якщо ($x_1 + j \bmod w < W_0$) і ($j \operatorname{div} w < H_0$)
            $b[(j \operatorname{div} w) \bmod h, j \bmod w] := C[j \operatorname{div} w, x_1 + j \bmod w]$;
        $j := j + M$;
    $y_1 := 0$;
    поки $y_1 + h < H_0$ { цикл по рядках області пошуку }
        $j := i$;
        поки $j < wh$ { завантаження рядків стовпця }
            якщо ($x_1 + j \bmod w < W_0$) і ($y_1 + j \operatorname{div} w < H_0$)
                $b[(y_1 + j \operatorname{div} w) \bmod h, j \bmod w] := C[y_1 + j \operatorname{div} w, x_1 + j \bmod w]$;
            $j := j + M$;
        виконати синхронізацію потоків; { дочекатися кінця завантаження області пошуку}
        $j := i$;
        поки $j < (w - W + 1)h$ { цикл по позиціях шаблону }
            $x_0 := j \bmod (w - W + 1)$; { ($x_0$, y_0) – координати позиції шаблону }
            $y_0 := j \operatorname{div} (w - W + 1)$; { всередині частини області пошуку у кеші }
            якщо ($y_0 + y_1 < H_s$) і ($x_1 + x_0 < W_s$)
                $s := 0$; { обчислення $D(x_0 + x_1, y_0 + y_1)$}
                    для y від 0 до H-1
                        p := посилання на $b[(y_0 + y_1 + y) \bmod h, x_0 + x_1]$;
                        для x від 0 до W-1
                            $s := s + Dist(f[y,x], p[x])$;
                    зберегти $D[y_0 + y_1, x_0 + x_1] = s$; { збереження результату }
                якщо $D_{best}^i > s$ { перевірка найкращого результату у потоці }
                    $D_{best}^i := s$; $x_{best}^i := x_0$; $y_{best}^i := y_0$;
            $j := j + M$;
        $y_1 := y_1 + h$; { перехід униз до наступної групи рядків }
        виконати синхронізацію потоків; { дочекатися закінчення обчислень перед початком завантаження наступної частини області пошуку }
    $x_1 := x_1 + w - W + 1$; { перехід управо до наступного стовпця }

Рисунок 6 – Алгоритм пошуку шаблону при розбитті області по рядках і стовпцях

Для алгоритму, що проводить розбиття по стовпцях і рядках, можлива модифікація: якщо ширина останнього стовпця не рівна ($W - 1$), то при його обробці можливо використовувати менше значення w і більше значення h, ніж для решти стовпців. Це може зменшити число кроків, необхідних для його обробки, внаслідок чого розшириться множина значень W_0, H_0, W і H, при яких алгоритм може працювати ефективно.

Практичні результати

Описані алгоритми реалізовано і протестовано на відеокарті GeForce 9800 GT. Результати порівнювалися з однопотоковою реалізацією на процесорі Athlon X2 5400.

Час пошуку 1000 фрагментів для запропонованих алгоритмів із використанням цілочисельної і дійсної арифметики приведено в табл. 1. Для CUDA наведено чистий час обчислень без урахування накладних витрат на передачу даних на відеокарту.

Таблиця 1 – Середній час пошуку 1000 фрагментів на CPU та GPU

Розмір шаблону	Розмір області пошуку	На центральному процесорі, мс	Час на CUDA з використанням текстурного кешу, мс		Час на CUDA з використанням спільної пам’яті, мс	
			цілочисельні	дійсні	цілочисельні	дійсні
8	39	328	13,3	9,4	11,8	10,6
8	71	1344	51,5	36,1	46,3	41,3
16	79	4860	196	130	167	149
16	143	19437	782	514	658	582

Як видно з табл. 1, використання відеокарти середнього рівня забезпечує прискорення в порівнянні з одним ядром процесора приблизно в 36 разів. При цьому знижується завантаження ЦП і він може паралельно проводити інші обчислення. При використанні двох або чотирьох ядер процесора час обробки множини шаблонів скоротиться майже в два або чотири рази; очікується, що при відеокарті рівня GeForce GTS 295 час обчислень на CUDA також скоротиться приблизно у чотири рази. На практиці прискорення може бути меншим через накладні витрати на підготовку даних у пам’яті пристрою.

При використанні цілочисельної арифметики алгоритм з явним кешуванням даних в спільній пам’яті, оптимізованим для даної задачі, виявився ефективнішим за алгоритм, що використовує апаратно реалізований текстурний кеш, навіть коли об’єм використовуваної спільної пам’яті був менший за об’єм текстурного кешу.

При приведенні даних до дійсного типу перед обчисленням (1) текстурний кеш виявився ефективнішим за рахунок швидкого перетворення цілого числа в нормалізоване дійсне, що є неможливим для даних в спільній пам’яті. Це співвідношення може змінитися в майбутніх пристроях на користь спільної пам’яті при прискоренні операцій цілочисельної арифметики або перетворення типів.

Висновки

У статті представлено апаратно-орієнтовані алгоритми пошуку шаблонів зображень довільної форми для паралельної архітектури CUDA. Вони можуть бути використані для підвищення швидкості відстежування об’єктів у відеопотоці за методом [2].

Алгоритми реалізовані і досліджені експериментально. В порівнянні з одним ядром процесора Athlon X2 5400 на відеокарті GeForce 9800 GT досягнуте зменшення часу пошуку множини фрагментів зображень до 36 разів. Використання пристрою з архітектурою CUDA може стати хорошою альтернативою використанню багатопроцесорних ЕОМ або кластеру при відстежуванні об’єктів у відеопотоці.

Досліджено кешування області пошуку у спільній пам’яті мультипроцесора та текстурному кеші. На сучасних пристроях CUDA доцільно використовувати текстурний кеш, але якщо у майбутніх пристроях буде прискорено виконання цілочисельних арифметичних операцій [7], для певних розмірів фрагменту та області пошуку може стати доцільним кешування області пошуку у спільній пам’яті.

При великому розмірі області пошуку більше прискорення за рахунок зниження точності можна отримати при реалізації швидких алгоритмів пошуку шаблонів [3]. При великому розмірі шаблону доцільно кешувати його у спільній пам’яті мультипроцесора по частинах.

Ще більше прискорення відстежування фрагментів при меншому енергоспоживанні і апаратних витратах можливе при реалізації на ПЛІС спеціалізованого векторного процесора для пошуку фрагментів зображень. При цьому можуть бути використані представлені в статті алгоритми кешування області пошуку.

Література

1. Cucchiara R. Track-based and object-based occlusion for people tracking refinement in indoor surveillance [Електронний ресурс] / R. Cucchiara, C. Grana, G. Tardini // Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks 2004. – New York, NY, USA, October 15, 2004. – 10 p. – Режим доступу : http://doi.acm.org/10.1145/ 1026799.1026814.
2. Ладиженський Ю.В. Відстежування об'єктів у відеопотоці на основі відстежування переміщення фрагментів об'єктів / Ю.В. Ладиженський, А.О. Середа // Наукові праці Донецького національного технічного університету. Серія: «Обчислювальна техніка та автоматизація». Випуск 17 (148). – Донецьк : ДонНТУ, 2009. – 215 с.
3. Adaptive Motion Estimation Processor for Autonomous Video Devices / T. Dias, S. Momcilovic, N. Roma, L. Sousa [Електронний ресурс]// EURASIP Journal on Embedded Systems. – 2007. – Article ID 57234. – P. 10. – Режим доступу : http://www. hindawi.com/GetArticle.spx?doi=10.1155/2007/57234.
4. Baker S. Lucas-Kanade 20 Years On: A Unifying Framework. / S. Baker, I. Matthews // Int. Journal of Computer Vision. – 2004. –Vol. 56, № 3, March. – P. 221-255. – Режим доступу: http://www.ri.cmu.edu/ pub_files/ pub3/baker_simon_2004_1/baker_simon_2004_1.pdf.
5. Sheu-Chih Cheng. A comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementation / Sheu-Chih Cheng, Hsueh-Ming Hang // IEEE Transactions on Circuits and Systems for Video Technology. – 1997. – Vol. 7, № 5. – P. 741-747.
6. NVIDIA CUDA Programming Guide Version 2.3.1, 2009. [Електронний ресурс]. – 145 p. – Режим доступу : http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf.
7. NVIDIA CUDA C Programming Best Practices Guide CUDA Toolkit 2.3, 2009. [Електронний ресурс]. – 145 p. – Режим доступу : http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_ CUDA_BestPracticesGuide_ 2.3pdf.

Ю.В. Ладыженский, А.А. Середа
Поиск фрагментов изображений по шаблону для отслеживания объектов в видео на параллельной архитектуре CUDA
Рассмотрен поиск фрагментов изображений произвольной формы по шаблону методом полного перебора с использованием параллельной вычислительной архитектуры CUDA. Предложены и проанализированы различные способы кэширования области поиска в памяти мультипроцессора. Экспериментально оценено ускорение по сравнению с центральным процессором. Предложенные алгоритмы могут использоваться для ускорения отслеживания объектов в видео.

Y. Ladyzhenskyy, A. Sereda
A Search of Image Fragments for Object Tracking in Video with Template Matching on the CUDA Parallel Architecture
A search of arbitrary shape image fragments with full-search template matching on CUDA is examined. Different approaches to search area caching in a multiprocessor's memory are proposed and analyzed. Acceleration on GPU in comparison to CPU is evaluated. The proposed algorithms can be used to accelerate object tracking in video.

Стаття надійшла до редакції 02.07.2010.